Explanation of Denormalization

The way that we will denormalize the database is by maintaining a name, and email, attribute both in personal and corporate account. To maintain data integrity when a value from one is created / changed, it will be checked against the other. As well, it is assumed that all corporate accounts will be paying with a card and cannot therefore be private.

Explanation of Denormalization

—Employees and chefs

For the different types of chefs we had their salary in each one as well as the hiredDate for each. Since all of these people share the same values as well as the Manager we moved them up to full time employee. However, we did not place the values in the values in the Employee’s tab as part time employees do not get paid a salary, since they are not full time.

— MaitreD

We intentionally left MaitreD blank because we split part time employees into 3 seperate jobs and the maitreD doesn’t have any attributes that make them unique but it is a title so we left Maitre D blank to make it easy to conceptualize and to query.

— Phone order and Web order

We intentionally left Phone order and Web order blank because the attributes that would be associated with the 2 classes can be found in other classes if we added those attributes, it would cause redundancy.

— Private Customer

We want to distinguish different types of customers. But the private customers don’t give out any information

— Appetizer

The only attribute that matters to appetizer so the name, but we can find it in the MenuItem class

— Orders and payments

For the payments we have only two classes the cash and card payment we could have split the credit and debit card, but by having them together we have easier access to the cards that were used.

— Cash

Added denomination to the cash values to count which bills are used for payment

— Recipe

For the recipe there are many names for them and each one can be different from another. Also, they all have their own unique ingredients to use that are multi value variables, and each as its own quantaity for creating the recipe. We can have places ingredients into its own class, but this would add an extra query whenever we wanted to see the recipe that a head chef had created. This allows us to have access to all the ingredients of a recipe all in one table.

— Payments

For the payments we have only two classes the cash and card payment. We decided to use denormalization for card payments as the only unique value between credit and debit are the respective types. This allows for easy query of all card payments, or isolation of one card payment type.

— Cash

Added denomination to the cash values to count which bills are used for payment. Separation of denomination would add complexity that is unnecessary. For this instance we don’t care about the denomination, but rather whether it is cash or card payment.

— Menu

Menu has originally four different menus evening, lunch, sunday brunch buffet, and children’s. Seperate classes could be used to separate them, but would make changing the menu price modifier difficult to use. One attribute that can be used to change the menu price modifier and is used to add redundancy when changing prices